Rajalakshmi Engineering College

Name: HARI PRASATH PALANIMURUGAN
Email: 241801080@rajalakshmi.edu.in
Roll no: 241801080
Phone: 8072956499
Branch: REC
Department: I AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 0

Section 1 : Coding

1. Problem Statement

Mike is learning about Binary Search Trees (BSTs) and wants to implement various operations on them. He wants to write a basic program for creating a BST, inserting nodes, and printing the tree in the pre-order traversal.

Write a program to help him solve this program.

Input Format

The first line of input consists of an integer N, representing the number of values to insert into the BST.

The second line consists of N space-separated integers, representing the values to insert into the BST.

Output Format

The output prints the space-separated values of the BST in the pre-order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
3 1 5 2 4

Output: 3 1 2 5 4

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->left = newNode->right = NULL;
   return newNode;
}
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

struct TreeNode {
   int data;
   struct TreeNode* left;
   struct TreeNode* right;
};

// Function to insert a new node into the BST
struct TreeNode* insert(struct TreeNode* root, int key) {
   if (root == NULL) {
       struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
       newNode->data = key;
       newNode->left = newNode->right = NULL;
       return newNode;
   }

   if (key < root->data) {
       root->left = insert(root->left, key);
   } else {
       root->right = insert(root->right, key);
   }

   return root;
}

// Function to find the minimum value node in a given BST
struct TreeNode* findMin(struct TreeNode* root) {
   while (root && root->left != NULL) {
       root = root->left;
   }
   return root;
}

// Function to delete a node with a given value from the BST
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
   if (root == NULL) {
       return root;
   }

   if (key < root->data) {
       root->left = deleteNode(root->left, key);
   } else if (key > root->data) {
       root->right = deleteNode(root->right, key);
   } else {
       // Node to be deleted found

       // Case 1: Node has no children (leaf node)
       if (root->left == NULL && root->right == NULL) {
           free(root);
           return NULL;
       }

       // Case 2: Node has only one child
       if (root->left == NULL) {
           struct TreeNode* temp = root->right;
           free(root);
           return temp;
       } else if (root->right == NULL) {
           struct TreeNode* temp = root->left;
           free(root);
           return temp;
       }

       // Case 3: Node has two children
       struct TreeNode* temp = findMin(root->right);
       root->data = temp->data;
       root->right = deleteNode(root->right, temp->data);
   }

   return root;
}

// Function to perform an in-order traversal of the BST
void inorderTraversal(struct TreeNode* root) {
   if (root != NULL) {
       inorderTraversal(root->left);
       printf("%d ", root->data);
       inorderTraversal(root->right);
   }
}

int main() {
   int N, rootValue, V;
   scanf("%d", &N);
   struct TreeNode* root = NULL;

   for (int i = 0; i < N; i++) {
       int key;
       scanf("%d", &key);
       if (i == 0) rootValue = key;
       root = insert(root, key);
   }
   scanf("%d", &V);
   root = deleteNode(root, V);
   inorderTraversal(root);

   return 0;
}

int main() {
   struct Node* root = NULL;

   int n;
   scanf("%d", &n);

   for (int i = 0; i < n; i++) {
       int value;
       scanf("%d", &value);
       root = insert(root, value);
   }

   printPreorder(root);
   return 0;
}
```

Status : Wrong ***Marks : 0/10***